PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/40, 9/34 — A1

(11) International Publication Number: WO 97/14101

(43) International Publication Date: 17 April 1997 (17.04.97)

(21) International Application Number: PCT/US96/15914

(22) International Filing Date: 3 October 1996 (03.10.96)

(30) Priority Data:
08/541,419 10 October 1995 (10.10.95) US

(71) Applicant (*for all designated States except US*): MICROUNITY SYSTEMS ENGINEERING, INC. [US/US]; 255 Caspian Drive, Sunnyvale, CA 94089 (US).

(72) Inventor; and

(75) Inventor/Applicant (*for US only*): HANSEN, Craig, C. [US/US]; 350 Yerba Santa Avenue, Los Altos, CA 94022 (US).

(74) Agent: PETERSON, James, W.; Burns, Doane, Swecker & Mathis, L.L.P., P.O. Box 1404, Alexandria, VA 22313-1404 (US).

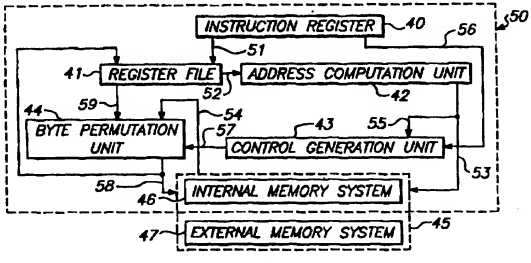
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND SYSTEM FOR FACILITATING BYTE ORDERING INTERFACING OF A COMPUTER SYSTEM



(57) Abstract

A method and data processing system for transferring data between the system and a memory system using more than one byte ordering convention by incorporating byte order information into instruction codes. The byte order information is coupled to a control unit along with other information characterizing the data transfer operation. In response to the byte order information and the data transfer operation information, the control unit generates a control signal that is coupled to a BPU. The control signal causes the BPU to rearrange the order of bytes in the data being transferred when the byte order information indicates a first byte ordering format. When the byte order information indicates a second byte ordering format, the BPU does not change the order of the bytes in the data being transferred.

METHOD AND SYSTEM FOR FACILITATING BYTE ORDERING INTERFACING OF A COMPUTER SYSTEM

FIELD OF THE INVENTION

The present invention relates to the field of computer systems, and more particularly to computer system byte ordering formats.

BACKGROUND OF THE INVENTION

In a computer system, information is transferred between devices within the system in the form of bits or bytes (i.e. eight bits) of binary data. In general, the data is transferred on buses that are capable of transmitting some multiple of eight bits, where the width of the bus indicates the number of bits it is capable of transmitting. Further, the amount of data that is transferred on a given bus may be the same width as the bus or less. For instance, a bus having a width of 64 bits (i.e. 8 bytes) may be used to transfer only one byte of data (i.e. 8 bits).

Data transfers between a computer system's central processing unit (CPU) and a memory system that stores digital data is one of the main data transfer operations performed in a computer system. The two types of data transfer operations that are performed between a CPU's registers and a memory system's memory locations are load and store operations. A load operation is the transfer of data from a designated memory location within a memory to a register within the CPU and a store operation is the transfer of data from the CPU's register to a designated memory location within a given memory.

In order to perform either a load or store operation the CPU provides an instruction that includes many different pieces of information. For instance, the instruction provides information that is used to determine which memory location (i.e. the address) that the load or store operation will involve. Specifically, the address information indicates the location within the memory in which the processor wants to store the data and the register in the CPU that the data is to be stored from (in the case of a store operation). Alternatively, the address information identifies the location within the memory where data is to

In the past, the manner in which the above described situation has been avoided is that computer systems are implemented with CPUs and devices using only one specific byte ordering scheme. There are many CPUs available that perform data transfers using either L.E. or B.E. byte ordering. For example, the PDP11 and VAX series manufactured by Digital Equipment Corporation (DEC), and the 86 series of microprocessors (such as the 80386, 80286, 8086, etc.) manufactured by Intel Corporation use L.E. byte ordering. Further, the IBM 360 and 370 series of CPUs as well as the Motorola 68000 series of microprocessors (such as the 68000, 68020, 68030, etc.) follow the B.E. byte ordering style.

One type of CPU described in U.S. Patent serial number 4,959,779 follows one byte ordering convention internally but is capable of converting ingoing and out-going data to adapt it to a selectable external ordering scheme. This CPU design adapts to the different byte ordering schemes by passing incoming data through a load aligner and out-going data through a store aligner. The load and store aligners are controlled by a shift converter. The shift amount converter, in response to the two lower order bits of the address of the data being transferred (along with other data transfer information), determines the amount of shift for the data being transferred and correspondingly alters the address of the data. In the case in which the internal byte ordering is the same as the external byte ordering, the data is not shifted. In the case in which the internal and external byte order format are different, the shift amount converter determines a shift amount to accommodate the difference in the internal and external byte ordering and renumbers the address of the data accordingly.

The main drawback of the apparatus described in U.S. Patent serial number 4,959,779 is that it assumes that the ordering scheme of the CPU is set to what is indicated in the status register of the CPU. In making this assumption, all store operations are performed such that data is stored in the byte ordering as indicated by the status register. Consequently, problems occur when the status register is subsequently changed to indicate a new byte ordering scheme.

In one embodiment, the processor instruction code includes information defining the direction and size of the data transfer operation being performed, a byte ordering indicator that defines the byte ordering format of the data to be transferred between the memory system and the processor, and address information utilized to perform the data transfer operation.

The address information incorporated in the instruction code is coupled to an address computation unit which functions to generate the addresses needed to perform the data transfer operation. A portion of the calculated addresses, the data size, and the byte ordering indicator, are coupled to a control generation unit that generates a control signal that is used to control a byte permutation unit. The byte permutation unit functions to either pass, shift, rearrange, or rearrange and shift the bytes of the transferred data depending on the status of the byte ordering indicator, the calculated addresses, and data size of the processor instruction.

In the method of the present invention, if the byte ordering scheme of the processor of the present invention and the memory system is the same, the byte ordering indicator in the instruction code is in a first state and the data is passed through the permutation unit without rearranging the order of the bytes.

In the case in which the processor and the memory system have different byte ordering schemes, the byte ordering indicator in the instruction code is in a different state other than the first state and the byte permutation unit rearranges the bytes of the transferred data in the manner specified by the byte order indicator. After the data is processed by the permutation unit it is transferred to a given location as specified by the instruction code. In the case in which a data transfer having a size less than the width of the processor bus is performed, data is also shifted by the permutation unit as designated by the instruction code. In one embodiment of the present invention, the processor is capable of performing parallel transfers of 16 bytes of data at one time. In a variation of the above embodiment, the permutation unit is capable of performing zero fill and sign extension operations as designated by the instruction code.

Figures 10A-10G illustrate Big and Little Endian load operations performed by a processor using L.E. byte ordering in accordance with one embodiment of the method and apparatus of the present invention.

Figure 11A illustrates a 4-byte L.E. store operation in accordance to the method and apparatus of the present invention having an address with an offset of zero.

Figure 11B illustrates a 4-byte B.E. store operation in accordance to the method and apparatus of the present invention having and address with an offset of 4.

DETAILED DESCRIPTION

A method and system for providing a means of performing data transfer operations between a processor and a memory system having more than one byte ordering format is described. In the following description, numerous specific details are set forth, such as data transfer size, bus size, instruction format, and processor operations in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that these specific details need not be employed to practice the present invention. In other instances, well-known processor structures and data and address accessing steps have not been described in detail in order to avoid unnecessarily obscuring the present invention.

Figures 1A and 1B illustrate a word of data having bytes b(0) - b(7) formatted using two different types of byte ordering; "Little Endian" (L.E.) and "Big Endian" (B.E.), respectively. Figure 1A illustrates bytes b(0) - b(7) stored in register 10 in L.E. byte ordering - where b(0) is the least significant byte (LSB) and b(7) is the most significant byte (MSB). As shown, each byte has an associated offset value ranging from 0 - 7. The offset of a byte is defined as the number of byte locations that a particular byte is displaced from a defined zero-offset point within a given storage word. The zero off-set point of the L.E. word is the LSB. For instance, as indicated in Figure 1A, b(0) is stored in the zero offset byte location and is said to have an associated off-set of 0. Further,

transferred are shifted a constant shift amount. Figure 2A shows an eight byte processor register 12 and an eight byte memory block 13 after a 4-byte L.E. load operation is performed having an associate address offset of 0. As shown, processor register 12 loads bytes b(0) - b(3) in little endian order, i.e. b(0) has an off-set of 0 and b(3) has an offset of 3 from the LSB. Since a L.E. load operation is being performed, the order of the bytes remain the same when transferred such that they retain their original associated offsets.

Figure 2B illustrates the prior art method as taught by U.S. Patent serial number 4,959,779 in which a 4-byte B.E. load operation is performed with an address offset of 0 by a processor using L.E. byte ordering. As shown in Figure 2B, bytes b(4) - b(7) are loaded in processor register 12. When a B.E. load operation is performed, bytes b(4) - b(7) are shifted such that b(7) is loaded into register byte 3, b(6) is loaded into register byte 2, b(5) is loaded into register byte 1, and b(4) is loaded into register byte 0. As can be seen, the prior art method shifts all of the bytes a fixed amount (in this case 4 bytes) such that the relative order of the bytes are always the same.

In comparing the order of the bytes stored in memory block 14 (Figure 2B) to the order of the bytes loaded in B.E. order shown in Figure 1B, it can be seen that the order of bytes b(4) - b(7) is not the same. Consequently, although the prior art method shifts the bytes to adapt to B.E. byte ordering, the actual order of the shifted data is inconsistent with what is expected by a processor or memory system using B.E. byte ordering. Due to this byte ordering inconsistency, problems may arise when the relative byte order of the transferred data is necessary for understanding the transferred data when it is subsequently read.

In addition, a system using both L.E. and B.E. memory transfers will exhibit inconsistencies because the offsets are adjusted. For example, a 4-byte word stored with a L.E. store at address zero is not loaded with a B.E. load at address zero. Instead, the B.E. load employs an inconsistent address of four indicating an offset of 0. Due to this byte order inconsistency, problems may

instruction code, such as that shown in Figures 5A and 5B, is loaded into instruction register 40.

Figures 5A and 5B illustrate two embodiments of instruction codes that may be loaded into instruction register 40 of the present invention for performing data transfers according to the present invention. Figure 5A illustrates an instruction code having four fields 60-63; each field containing different encoded information. Field 60 contains 8 bits of data corresponding to the operation code. In one embodiment, the operation code provides three pieces of information. First it indicates the operation to be performed. For example, either a load or a store operation. The operation code also indicates the byte ordering format of the data transfer being performed such as L.E. or B.E. And finally, the operation code indicates the size of the data that is being transferred. Fields 61, 62, and 63 (i.e. ra, rb, and immediate) contain address information that is utilized for determining where the data is to be loaded or stored.

Figure 5B illustrates a second embodiment of a data transfer instruction code of the present invention containing 5 fields, 64 - 68. Fields 64 and 68 indicate the operation to be performed. Two operation code fields are utilized in this instruction code so as to increase the total potential number of instructions of the instruction set. For instance, the major operation code indicates a particular class of operation to be performed while the minor operation code indicates the particular operation within that class of operation. The minor operation code also indicates that size of the data transfer and the byte ordering format. Fields 65, 66, and 67 identify source and destination address locations for the data transfer. It should be understood that, the instruction code of the present invention requires that the byte order information, data size, and operation and address information be incorporated into it. However it should be obvious that the instruction code of the present invention is not limited to the number of bytes and format shown in Figures 5A and 5B and that other additional information may be included within the instruction code if desired.

computed address from bus 55, to generate a set of control signals on bus 57. The control signals on bus 57 are coupled to BPU 44.

BPU 44, in response to the control signal on bus 57, functions to either: 1) rearrange the bytes in the data being transferred when the byte order format indicated in instruction register 40 is other than the byte order format being utilized by processor 50 and the offset of the address is zero. 2) rearrange and shift the bytes of data being transferred when the offset of the address is non-zero and the byte order format indicated in instruction register 40 is other than the byte order format being utilized by processor 50, 3) pass the data when byte order of operation is the same as that being utilized by processor 50 and the offset of the address is zero, or 4) shift the data when byte order of operation is the same as that being utilized by processor 50 and the offset of the address is non-zero.

Once processed by BPU 44, the data is coupled back to register file 41 on bus 58 where it is loaded into the processor register as indicated by the destination address in the instruction code.

A store operation is performed in a similar manner as a load operation. Address information from the instruction register 40 is coupled to register file 41 and then to address computation unit 42. Unit 42 computes the necessary address information and couples it to CGU 43 along with operation information coupled on bus 56. CGU provides the appropriate control (on bus 57) to BPU 44. In response to control signals on bus 57, BPU 44 either rearranges, rearranges/shifts, shifts or passes data provided by register file 41 on bus 59. This processed data is then coupled to memory system 45 on bus 58 and stored into the memory location as indicated by the instruction.

In one embodiment of the present invention, processor 50 internally performs data transfers using L.E. byte order format. As a result, in the case of a B.E. load or store operation with memory system 45, data is either rearranged or rearranged and shifted. And, in the case of a L.E. load or store operation with memory system 45, data is either passed directly through the BPU or is shifted.

MAJOR	OPERATION
156	
157	
158	
159	L.MINOR

MAJOR	OPERATION
188	
189	
190	
191	S.MINOR

Minor Operation Code Field Values for L.MINOR
Table 2

L.MINOR	OPERATION
0	LU16LA
1	LU16BA
2	LU16L
3	LU16B
4	LU32LA
5	LU32BA
6	LU32L
7	LU32B
8	L16LA
9	L16BA
10	L16L
11	L16B
12	L32LA

L.MINOR	OPERATION
13	L32BA
14	L32L
15	L32B
16	L64LA
17	L64BA
18	L64L
19	L64B
20	L128LA
21	L128BA
22	L128L
23	L128B
24	L8
25	LU8

Minor Operation Code Field Values for S.MINOR
Table 3

S.MINOR	OPERATION
0	SAAS64LA
1	SAAS64BA
2	SCAS64LA
3	SCAS64BA
4	SMAS64LA
5	SMAS64BA
6	SMUX64LA
7	SMUX64BA
8	S16LA
9	S16BA
10	S16L
11	S16B
12	S32LA

S.MINOR	OPERATIO
13	S32BA
14	S32L
15	S32B
16	S64LA
17	S64BA
18	S64L
19	S64B
20	S128LA
21	S128BA
22	S128L
23	S128B
24	S8

(Note: all instructions and buses are shown in little endian format).

CODE	INSTRUCTION	OP	SIGN	SIZE	ORDER	ALIGN
5	LU32BA	L	U	32	B	A
6	LU32L	L	U	32	L	u
7	LU32B	L	U	32	B	u
8	L16LA	L	s	16	L	A
9	L16BA	L	s	16	B	A
10	L16L	L	s	16	L	u
11	L16B	L	s	16	B	u
12	L32LA	L	s	32	L	A
13	L32BA	L	s	32	B	A
14	L32L	L	s	32	L	u
15	L32B	L	s	32	B	u
16	L64LA	L	-	64	L	A
17	L64BA	L	-	64	B	A
18	L64L	L	-	64	L	u
19	L64B	L	-	64	B	u
20	L128LA	L	-	128	L	A
21	L128BA	L	-	128	B	A
22	L128L	L	-	128	L	u
23	L128B	L	-	128	B	u
24	L8	L	s	8	-	-
25	LU8	L	U	8	-	-
26		-	-	-	-	-
27		-	-	-	-	-
28		-	-	-	-	-
29		-	-	-	-	-
30		-	-	-	-	-
31		-	-	-	-	-
32	SAAS64LA	SAAS	-	64	L	A
33	SAAS64BA	SAAS	-	64	B	A
34	SCAS64LA	SCAS	-	64	L	A
35	SCAS64BA	SCAS	-	64	B	A
36	SMAS64LA	SMAS	-	64	L	A
37	SMAS64BA	SMAS	-	64	B	A
38	SMUX64LA	SMUX	-	64	L	A
39	SMUX64BA	SMUX	-	64	B	A
40	S16LA	S	-	16	L	A
41	S16BA	S	-	16	B	A
42	S16L	S	-	16	L	u
43	S16B	S	-	16	B	u
44	S32LA	S	-	32	L	A
45	S32BA	S	-	32	B	A
46	S32L	S	-	32	L	u
47	S32B	S	-	32	B	u
48	S64LA	S	-	64	L	A
49	S64BA	S	-	64	B	A
50	S64L	S	-	64	L	u

The multiplexer control generated by Mux Control 91 is determined as shown in Table 5 for each of the possible operation conditions:

Table 5

OPERATION CONDITION	OP	ORDER	MUX BITS 4*i+3..4*i
little-load	L	L	address $_{3..0}$ + i
big-load	L	B	address $_{3..0}$ + size $_{6..3}$ - i -1
little-swap	SAAS	L	
big-swap	SAAS	B	
little-swap	SCAS	L	
big-swap	SCAS	B	
little-swap	SMAS	L	
big-swap	SMAS	B	
little-store	SMUX	L	i - address $_{3..0}$
big-store	SMUX	B	size $_{6..3}$ - 1 - i address $_{3..0}$
little-store	S	L	i - address $_{3..0}$
big-store	S	B	size $_{6..3}$ - 1 - i address $_{3..0}$

(Note: address $_{3..0}$ is the lower order four bits of the byte address provided on bus 55, Figure 4, size $_{6..3}$ is the size of the operand in bytes, and all adds and subtracts in Table 5 are performed module 16; i.e. arithmetic is 4 bits wide with overflows discarded.)

Table 5 gives the mux control bits in general terms. For example, for the little-load operation in Table 5 in which op = L and order = L, mux control bits b3..b0 is address $_{3..0}$; mux control bits b7..b4 is 1 + address $_{3..0}$; mux control bits b11..b8 is 2 + address $_{3..0}$, etc. Thus the general form for determining the mux control bits is: for i:= 0 to 15; mux control bits 4*i+3..4*i is (address $_{3..0}$ + i). Similarly, for big-load (i.e. op = L and order = B), mux control bits b3..b0 is address $_{3..0}$ + size $_{6..3}$ - 1, mux control bits b7..b4 is address $_{3..0}$ + size $_{6..3}$ - 2; mux control bits b11..b8 is address $_{3..0}$ + size $_{6..3}$ - 3 etc. Consequently, the general form for the mux bits in this operation are: for i:= 0 to 15; mux control bits 4*i+3..4*i is (address $_{3..0}$ + size $_{6..3}$ - i - 1). In the case of operations SAAS (i.e. Little-swap) and SCAS (Big-swap) these operations are performed in two-cycles and both involve loads and stores; the

(MLU 95) and Sign Extension Unit 96 (SEU 96). The dataword to be processed by BPU 44 originates from either memory system 45 on bus 54 (in the case of a load operation) or from register file 41 on bus 59 (in the case of a store operation). In response to the "op" signal, multiplexer 94 passes data to MLU 95 from either bus 54 or bus 59. MLU 95 in response to the 64-bit mux control provided by mux control 91, rearranges the bits in the dataword as determined by the byte order provided in the mux control signals (as indicated in the instruction on bus 56).

The output of MLU 95 is coupled to Sign Extension Unit (SEU) 96 which functions to sign extend the dataword provided by MLU 95 according to the control signals "octlet", "quadlet", and "doublet" as determined by the "op", "sign", and "size" control signals.

Figure 8 illustrates one implementation of MLU 95 (for processing a dataword having up to 128 bits). It includes 128 16-to-1 multiplexers - each controlled by four bits of the 64-bit mux control signal provided by Mux Control 91. The 128-bit data input bus, DataIN(127. . .0), and the 64-bit control mux signal, mux control(63. . .0) are coupled to mux(127. . .0). Each of mux(127. . .0) outputs a single bit of data in the particular byte order as indicated by the instruction code onto one of 128-bit output bus DataOUT(127. . .0) . The rearranged dataword is then coupled to SEU 96.

Figure 9A-9C illustrate one implementation of SEU 96. In this implementation a 2, 4, or 8 byte sign extension can be performed on bits b63 - b0. Since no sign extension is performed on bits b127 - b64, they are passed directly to output bus 58 (Figure 6). Further bits b7 - b0 are passed directly to output bus 58. Bits b15 - b8, and b7 are coupled to the circuit as shown in Figure 8A which functions to fill bit locations b15 - b8 with either: 1) the b7 value (in the case of a 2-byte sign extend), 2) the actual value of the bit (in the case of no sign extend), or 3) with a "0" (in the case of a zero fill) in response to the doublet control signal. Similarly, Figure 9B functions to perform the sign extension operation on bits b16 - b31 in response to the quadlet control signal by filling these bit locations with either: 1) the b7 value (in the case of a 2 byte

Figure 10E illustrates an 8-byte B.E. load operation from memory area 71 to register 76 in which all bytes are rearranged by BPU 44. Figure 6F illustrates a single byte B.E. and L.E. load operation between a memory area 71 and processor register 77. As described above, this operation is the same for both B.E. and L.E. operations since a single byte is consistently loaded into the same byte location for both a B.E. single byte load and a L.E. single byte load. Consequently, a single instruction can be used to perform both of these operations.

The present invention also applies to unaligned data transfers in which the memory address in the transfer instruction corresponds to a number which is not a multiple of the size of the memory transfer. Figure 10G illustrates an example of a 4-byte unaligned B.E. load operation according to the method of the present invention from memory area 71 to processor register 78 where the address indicated in the instruction corresponds to byte location b(2). It should be noted that some unaligned transfers cross memory word boundaries and therefore reference two adjacent memory words.

Also illustrated (in Figures 11A and 11B) are examples of two store operations performed in accordance to the method and system of the present invention. Figure 11A illustrates a 4-byte L.E. store operation having an address with an offset of zero. In this operation, bytes b(0) - b(3) in processor register 80 are transferred to memory location 81 in the same order with an offset of zero. Figure 11B illustrates a 4-byte B.E. store operation having an address with an offset of 4 in which bytes b(0) - b(3) are both rearranged and shifted (by 4 bytes) during the transfer of data between processor register 80 and memory location 82.

It should be noted that in some cases it is advantageous to standardize the byte ordering of data that is interpreted directly by the processor and is larger than a single byte (such as processor instructions that may be employed interchangeably within both L.E. software and B.E. software applications). Standardizing this type of data eliminates the potential of misinterpreting instructions. Thus, in one embodiment of the present invention, L.E. byte

CLAIMS

I Claim:

1) A method of transferring data between a first storage area within a system and a second storage area, wherein said data comprises a given number of bytes, said bytes having an associated order, said system being controlled by encoded instructions and said instructions indicating a data transfer operation defining an associated address offset within said second storage area, said method comprising the steps of:

incorporating byte order information in said instructions, said byte order information indicating a byte ordering format;

rearranging said order of said bytes during said data transfer operation if said byte order information indicates a first byte order format;

leaving said order of said bytes the same during said data transfer operation if said byte order information indicates a second byte ordering format.

2) The method as described in claim 1 wherein said first and second byte ordering formats are one of the "Little Endian" and "Big Endian" byte ordering formats.

3) The method as described in claim 2 further including the step of shifting said data during said data transfer when said offset is non-zero.

4) The method as described in claim 3 wherein said first storage area is a processor register and said second storage area is a memory location.

5) An encoded instruction executed by a data processing system comprising:

a first portion defining a transfer operation of a given number of bytes of data between a first storage area in said system and a second storage area;

a second portion providing address information utilized in said data transfer operation;

a third portion indicating a byte ordering format of said data.

6) The instruction as described in claim 5 wherein said byte ordering format is one of the "Little Endian" and "Big Endian" byte ordering formats.

12) The method as described in claim 10 or 11 wherein said first and second byte order are one of the "Little Endian" and "Big Endian" byte order.

13) The method as described in claim 12 wherein said first storage area is a processor register and said second storage area is a memory location.

14) A set of encoded instructions for a given data transfer operation for use in said system which performs data transfer operations between a first storage area in said system and a second storage area, wherein said data comprises a given number of bytes, said bytes of data having an associated order, said system being controlled by encoded instructions, said set of instructions comprising:

a set of multiple byte data transfer instructions corresponding to said given data transfer operation when said data has multiple bytes, each instruction in said set of multiple byte instructions specifying a distinct byte order, where said order of said multiple bytes depends upon said specified byte order of said each multiple byte instruction;

a set of single byte instructions corresponding to said given data transfer operation when said data has a single byte, each instruction in said set of single byte instructions having an associated distinct byte order, where said single byte is stored into said second storage area independent of said associated distinct byte order of said each single byte instruction.

15) The set of instructions as described in Claim 14 wherein said each set of single byte instructions and said each set of multiple byte instructions includes a first instruction specifying a first byte ordering and a second instruction specifying a second byte ordering.

16) The set of instructions as described in Claim 14 wherein said each set of single byte instructions includes one instruction that does not specify said byte order and wherein said each set of multiple byte instructions includes a first instruction specifying a first byte ordering and a second instruction specifying a second byte ordering.

said single byte transfer operations are performed and said single byte is stored into said second storage area independent of said byte order information.

22) The system as described in claim 20 or 21 wherein said instructions include information indicating said number of said bytes.

23) The system as described in claim 22 wherein said first and second byte ordering formats are one of the "Little Endian" and "Big Endian" byte ordering formats.

24) The system as described in claim 23 wherein said first storage area is a processor register and said second storage area is a memory location.

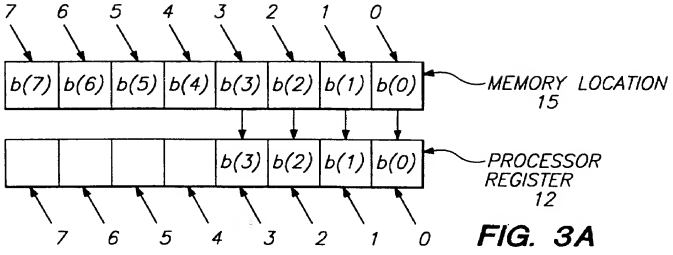


FIG. 3A

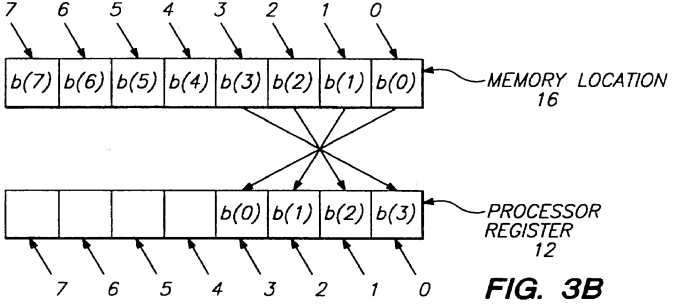


FIG. 3B

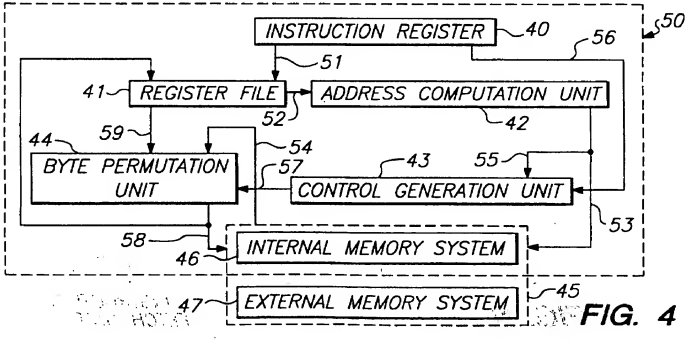


FIG. 4

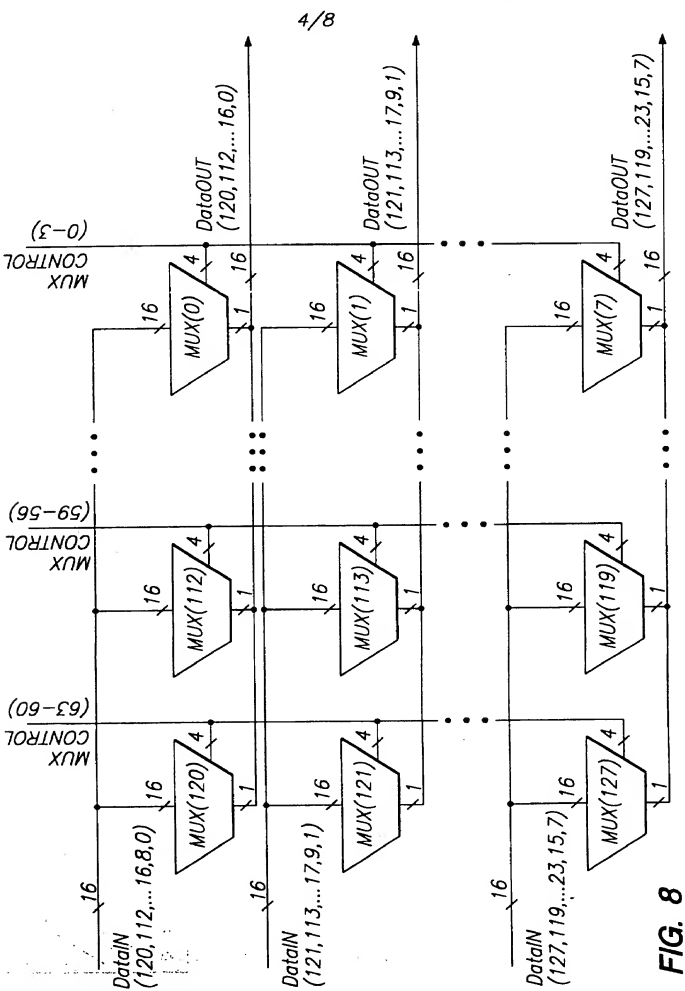

DataIN (120,112,...16,8,0)
DataIN (121,113,...17,9,1)
DataIN (127,119,...23,15,7)
16
MUX CONTROL (63-60)
MUX CONTROL (59-56)
MUX CONTROL (0-3)
MUX(120)
MUX(121)
MUX(127)
MUX(112)
MUX(113)
MUX(119)
MUX(0)
MUX(1)
MUX(7)
4
1
DataOUT (120,112,...16,0)
DataOUT (121,113,...17,9,1)
DataOUT (127,119,...23,15,7)

FIG. 8

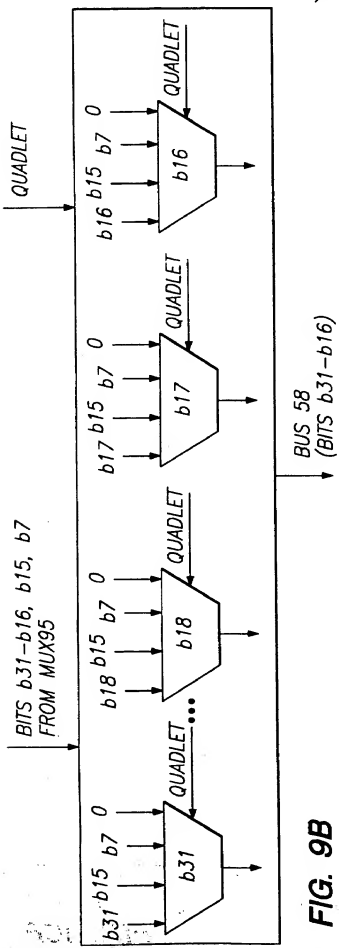

QUADLET
BITS b31–b16, b15, b7
FROM MUX95
b31 b15 b7 0
b18 b15 b7 0
b17 b15 b7 0
b16 b15 b7 0
QUADLET
b31
b18
b17
b16
BUS 58
(BITS b31–b16)

FIG. 9B

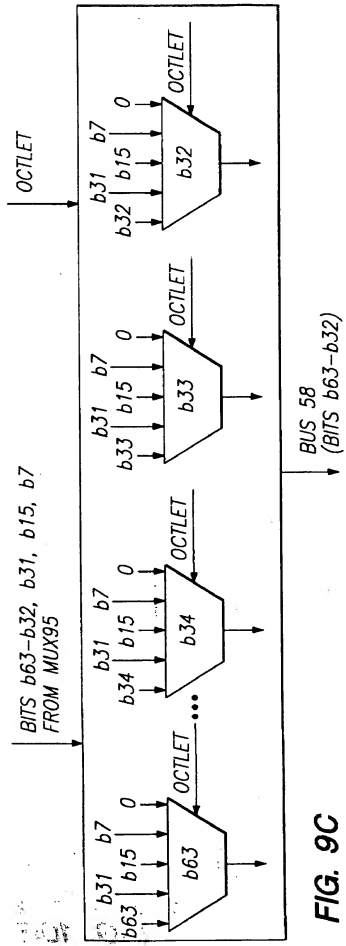

OCTLET
BITS b63–b32, b31, b15, b7
FROM MUX95
b31 b7
b63 b15 0
b34 b15 0
b33 b15 0
b32 b15 0
OCTLET
b63
b34
b33
b32
BUS 58
(BITS b63–b32)

FIG. 9C

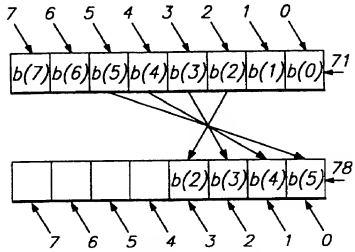


FIG. 10G

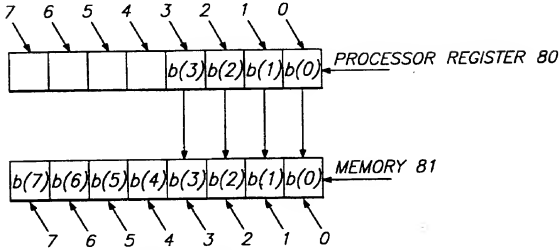


FIG. 11A

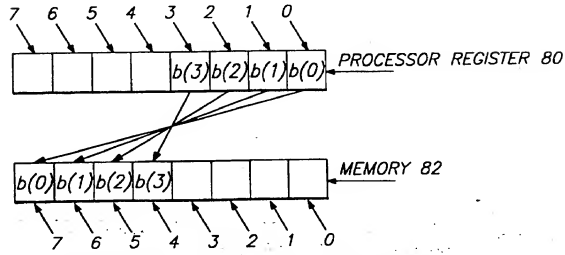


FIG. 11B

INTERNATIONAL SEARCH REPORT

information on patent family members

Inter nal Application No
PCT/US 96/15914

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
GB-A-2229832	03-10-90	DE-A-	4010119	04-10-90
		FR-A-	2645293	05-10-90
		HK-A-	107293	22-10-93
		IT-B-	1239828	15-11-93
		JP-A-	2285426	22-11-90
WO-A-9415269	07-07-94	EP-A-	0629303	21-12-94
		JP-T-	7505972	29-06-95
US-A-4814976	21-03-89	AU-B-	619734	06-02-92
		AU-A-	1185288	15-07-88
		CA-A-	1293331	17-12-91
		JP-T-	1502700	14-09-89
		KR-B-	9603046	04-03-96
		WO-A-	8804806	30-06-88